TITLE OF THE INVENTION

NEGATIVE CACHE MANAGEMENT SYSTEM AND METHOD

COPYRIGHT NOTIFICATION

BACKGROUND OF THE INVENTION

1.   Field of the Invention

This invention relates to data processing systems. More particularly this invention provides a negative cache management system and method for controlling valid and invalid memory component identifiers, and, thereby, accelerates performance of relatively high speed memory components, including hardware and software systems, that operate with storage devices.

2.   Background

Data processing systems that are coupled to a large amount of data storage may spend significant amounts of CPU and I/O BUS time reading and writing data to and from data storage devices, such as disk drives or similar longer term data storage devices. Accordingly, many data processing systems include relatively high speed, short term caching systems that can be accessed much more rapidly by a coupled computer system without the delays associated with the mechanical motion and time delay inherent in a long term storage device. Conventional data processing systems may include caches at several levels of operation, such as, cache memory, disk cache, RAM, "macro" caches, local server caches, and web browser caches.

One of the most well known caching systems is the use of a name cache to access a directory or a file stored through the use of tree structured directories. Each disk contains a root directory, one or more subdirectories, and one or more files. Each directory contains

5   entries for additional directories and files. A named file may be at the end of a path that passes through a root directory and one or more subdirectories (i.e., /Directory/Subdirectory/Filename). A specific file may be identified by accessing the drive, path, and filename. When a user requests a specific file, the operating system file system must take the file's pathname and search through all of the components (i.e., directories and

10   subdirectories) to identify the location of the file associated with that filename.

In conventional file systems, the file system typically maintains a name cache containing the location of files that have earlier been requested and found. This avoids having to search again for a file every time that file is accessed. Traditionally, the unified

15   file manager (UFM) of the file system receives a pathname/filename and first looks in the name cache to see if the location of the requested file identifier is already there. If not, the file identifier is passed to the flat file manager (FFM) that starts searching for the file in the directory. Some users might have thousands, tens of thousands, or even conceivably more files in a single directory, making it potentially very time consuming to search.

20

If an invalid file identifier is received, the file system will search through all directories in the PATH environment variable without finding it. Since the requested file identifier was not successfully located, nothing gets placed in the name cache. If users frequently ask for the same invalid file identifiers, the file system will be forced to repeatedly

25   do useless searching that takes up valuable and CPU cycles and disk I/O. Accordingly, what is needed is a system and method for accelerating performance of relatively high speed memory components in such a way as to ensure that the memory addresses of invalid memory component identifiers are managed by a cache system and method that may be utilized by a coupled host computer in a relatively short period of time.

SUMMARY OF THE INVENTION

This invention provides an intelligent caching system and method that immediately accelerate the response time and throughput of all existing relatively high speed memory components, such as storage systems that include disk drives and arrays, connected to either single-, dual-, or multi- host computers. To achieve this, the caching system and method maintain one or more caches in memory that manage valid and invalid memory component identifiers, such as, caches of recently searched valid and invalid file and object identifiers, including directory files, as well as other miscellaneous information, that are used to locate directories and files in storage devices.

The caching system and method are transparent to the host computer and have an easily configurable cache capacity. In an embodiment, a negative caching system and method manages only the invalid directories and files. The negative cache management system and method can be configured as a predetermined percentage of capacity, such as 1% of a caching system and method populated with valid memory component identifiers. Alternatively, a predetermined finite number of spaces can be set aside for the negative cache. A separate negative cache is added to keep invalid memory component identifiers. This negative name cache, like the name cache, searches for a memory component identifier, such as a specified filename, when a memory component identifier is first received. If the name of the memory component identifier is stored in the negative name cache, the operating system knows that no further searching is required.

As an alternative, invalid memory component identifiers could be entered into the same name cache as the valid memory component identifiers. In this single cache embodiment, valid and invalid memory component identifiers can be differentiated by adding a validity field to the cache entry or, alternatively, by placing a default entry indicating invalidity in the field that would normally contain the location of the data fields. Similar to the embodiment that utilizes a separate negative cache, this single cache could be partitioned

and configured to store a predetermined percentage of valid, such as 99% of capacity, and a predetermined percentage, such as 1% of capacity, of invalid memory component identifiers. Alternatively, a predetermined finite number of spaces can be set aside for the invalid memory component identifiers.

5

In another embodiment, this invention includes (a) logic to check the cache to determine if a received memory component identifier is invalid, (b) logic to add the invalid memory component identifier to the cache when a search by the flat file manager is unsuccessful, (c) logic to monitor new memory component identifiers being created and to

10   remove an entry from the invalid list if that path name later becomes valid, (d) logic to monitor memory component identifiers being deleted or moved and to add an entry from the invalid list if that path name becomes invalid, (e) logic to manage the most frequently used invalid memory component identifiers, and (f) logic to manage the most recently used invalid memory component identifiers.

15

This invention is useful in providing a system and method for rapidly identifying an invalid memory component identifier, such as a directory/filename, after such memory component identifier is initially searched for and determined to be invalid.

20   Another use of this invention is to provide a system and method for rapidly identifying an invalid memory component identifier, such as a directory/filename, after such memory component identifier is deleted from or moved to another location of the storage device.

25   Another use of this invention is to reduce the time required to locate an invalid memory component identifier by accessing a negative cache that manages invalid memory component identifiers, and, therefore, eliminates the necessity of searching through the directories and filenames of a storage device, such as a disk drive.

Another use is to provide a system and method for rapidly identifying invalid memory component identifiers that have been recently searched for and that are frequently searched for. This is achieved by creating and maintaining an intelligent negative cache that uses the history of searches for invalid memory component identifiers.

5

These uses may be accomplished singularly, or in combination, in one or more of the embodiments of the present invention.

Additional uses, objects, advantages, and novel features of the invention is set forth in
10   the detailed description that follows and becomes more apparent to those skilled in the art upon examination of the following or upon learning by practice of the invention.

BRIEF DESCRIPTION OF THE FIGURES

Other objects, advantages, and novel features of the present invention are more clearly
15   understood by reference to the following description taken in connection with the accompanying figures, in which:

FIG. 1 is a representation of a data storage directory used in data processing systems.

20   FIG. 2 is a flowchart illustrating PATH usage in data processing systems.

FIG. 3 is a prior art representation of the memory components of a data processing system.

25   FIG. 4 is a flowchart illustrating the prior art method of searching for a filename in a data processing system.

FIG. 5 is a representation of the memory components of a data processing system including use of a negative name cache in an embodiment of the present invention.

FIG. 6 is a flowchart illustrating the method of searching for a memory component

5   identifier in a data processing system that includes use of a negative name cache in an embodiment of the present invention.

DETAILED DESCRIPTION

10   Referring now in detail to an embodiment of the invention, an example of which is illustrated in the accompanying attachments, the present invention makes use of two basic concepts. One concept is creating a cache system and method that contain valid and invalid memory component identifiers. For example, in response to an initial request to locate a specified filename, an operating system searches for data stored in a directory structure of a

15   storage device, such as a disk drive. If the specified filename is successfully located, it is added to a name cache in main memory. If the specified filename is not successfully located, it is added to a negative name cache in main memory. Thus, the present invention is different from conventional caching systems and methods in that it creates a negative cache for memory component identifiers that are not actually located on the storage device.

20

The second concept manages the negative cache based on the memory components coupled to the host computer. For example, in a disk caching system, a negative name cache is maintained with the unified file manager. The next time a search is requested for the specified file identifier, the parent directory and the filename stored in the negative name

25   cache can be immediately accessed by a unified file manager to determine whether the specified filename is stored on the disk. Thus, the present invention provides a negative cache management system and method that can be used to accelerate the input/output performance of various relatively high speed memory components, such as disk drives, arrays, routers, and buffers, connected to either single-, dual-, or multi- host computers.

Turning now to the attached figures, Fig. 1 depicts a conventional file system that is rooted at directory called "/" 100 and is referred to as the root directory. The root directory 100 has subdirectories "usr" 110, "tmp" 120, and "etc" 130 branching off of it. Each of these

5 subdirectories directories 110-130 contains files 121-122 or other subdirectories 111, 131. The UNIX "ls" command executable file 112 resides at "/usr/bin" 111 and its pathname is referred to as "/usr/bin/ls" 112. When a user types the command "ls," the system consults a special environment variable that instructs the system where "ls" could possibly be located. That variable is called a PATH environment variable, and an example of a PATH

10 environment variable is provided below.

PATH= /tmp : /etc/bin : /usr/bin

This PATH statement tells the system that whenever an executable file is searched for, the system should look in "/tmp" 120 first. If not found in "/tmp" 120 then look in

15 "/etc/bin" 131 next and finally, if still not found, look in "/usr/bin" 111. As it turns out in our scenario, "ls" happens to reside in the last directory, "/usr/bin" 111. This algorithm is sketched in Figure 2.

20 Fig. 2 illustrates the flowchart of PATH usage. When a request is received to execute command "ls," 201, the system consults the PATH environment variable 202 and gets the next component of the PATH 203. Thereafter, the system concatenates the component of PATH to "ls," 204 and looks up "ls" in the file system 205. The system queries if the file is found 206. If so, then the system executes the command 207. However, if the file is not

25 found, the system gets the next component of PATH 203 until the file is found.

The problem with this system is that if "ls" is a frequently invoked command that is not found in the "/tmp" directory, then the system is going to be frequently searching for it and wasting time. The system is always going to start the search at "/tmp" and this is always

going to fail. In some systems, the "/tmp" directory contains many thousands of files and searching through such a huge directory is unacceptably slow.

5   Figs. 3 and 4 depicts a conventional UFM subsystem that receives a directory pointer and a file name to look for in that directory. In this illustrative example, the directory is "/tmp" and the file name is "ls." To speed such frequent searches, the UFM 302 uses a Name Cache 303 that contains recent successful search results. However, an entry for "/tmp/ls" is not going to exist in the name cache because no such file exists in "/tmp." The Name Cache 303 remembers only successful results, not failures. In the case of frequent failures, the

10 Name Cache 303 is of no help, and the Name Cache 303 actually decelerates the processing system because it spends CPU cycles trying to locate something that does not exist. After failing to find "/tmp/ls" in the Name Cache 303, the system moves on and ask the FFM 305 to go look for "/tmp/ls." FFM 305 asks the Buffer Manager (BM) 306 to provide a buffer containing a disk copy of the directory contents. This step may involve disk Input/Output

15 (I/O) 308. The FFM 305 then traverses the directory contents one file at a time until the file is found or the end of the directory has been reached. This step involves accessing the Block Special File Manager (BSFM) 307 and accessing disk Input/Output 308. In the case of a huge "/tmp" directory, the system must inspect each one of its thousands of files before all entries are exhausted to declare failure. The problem with this search is that the system

20 doesn't remember the results of that painful computation and it throws it away. The next time the "ls" command is invoked, the system repeats that expensive failure sequence of computations. Figure 4 is a flowchart describing the above algorithm and illustrates how conventional name caches are populated.

25   Figs. 5 and 6 depict the UFM subsystem of the present invention including the negative name cache that receives a directory pointer and a file name to look for in that directory. In this illustrative embodiment, the directory is "/tmp" and the file name is "ls." To speed such frequent searches, the UFM 302 uses a both a Name Cache 303 that contains recent successful search results and a Negative Name Cache 503 that contains recent

unsuccessful search results. In this second illustrative embodiment, an entry for "/tmp/ls" is going to exist in the Negative Name Cache 503 because no such file exists in "/tmp" as established by a directory/filename search for "tmp/ls." The Negative Name Cache 503 remembers the failed attempt to locate an entry for "/tmp/ls," and the system accelerates its

5   CPU time by immediately identifying the invalid search request and avoiding the repeated computation used by conventional systems. Figure 6 is a flowchart describing the above algorithm and illustrate how an embodiment of the negative name cache is populated.

An example of the advantages of the negative name caching system is shown in Table

10   1 below for an exemplary storage device with two processors and a fileset that consists of 50 file systems.

	CONVENTIONAL CACHING SYSTEM	NEGATIVE CACHING SYSTEM
Files	1950050	1950050
Files accessed for I/O operations	195000	195000
Directories	65000	65000
Filename lookups/second when 3% of files are invalid	4000	5000

The above files are spread over the 50 file systems with a random attempt to lookup filename

15   in the system where 3% of the named files are invalid. The above results show improved CPU and I/O performance of 25%.

The present invention may be implemented in numerous embodiments. In one embodiment, a separate negative name cache is added to keep invalid file identifiers (e.g..,

20   Directory/Subdirectory/InvalidFilename). This cache, like the name cache, is searched when a file identifier is first received. If the name appears in the negative name cache, the operating system knows that no further searching is required. This prevent the need to search any further and results in a very significant performance boost.

Negative Cache Management System and Method

9

In another embodiment, invalid file identifiers could be entered into the same name cache as the valid file identifiers. In this single cache implementation, valid and invalid file identifiers could be differentiated by inspecting the inode pointer embedded in the cache entry. The inode pointer is a file system structure that contains information about a single

5 file. If the inode pointer is NULL, then the system identifies the file identifier as an invalid file that does not reside in the given directory. If the inode pointer is valid, then the system operates like a conventional name cache entry representing a success.

In another embodiment, this invention includes (a) logic to check the cache to

10 determine if a received memory component identifier is invalid, (b) logic to add the invalid memory component identifier to the cache, such as adding a file identifier to the cache when a search in a directory for a file name is unsuccessful, (c) logic to monitor new memory component identifiers being created and to remove an entry from the invalid list if that file identifier later becomes valid, (d) logic to monitor memory component identifiers being

15 deleted or moved and to add an entry from the invalid list if that file identifier becomes invalid, (e) logic to manage the most frequently used invalid memory component identifiers, and (f) logic to manage the most recently used invalid memory component identifiers.

In another embodiment, the negative cache size is variable, as it could have different

20 size requirements. For example, the size of the negative cache may be based on the usage of the system or based on a predetermined percentage of capacity, such as 1% of the conventional caching system and method populated with valid memory component identifiers. Another example is to set aside a predetermined finite number of spaces for the negative cache. In addition, the negative cache may be initialized at the same time as the

25 conventional cache during system initialization (i.e., boot time).

In another embodiment, the negative cache can be implemented by a number of standard cache implementation techniques. For example, when the name cache is constructed it initially contains no entries, positive nor negative. Thereafter, when a file

lookup operation results in failure, FFM will report back to UFM about the failure. UFM, in

turn, will create a new negative file identifier and place it as an entry to the negative name

cache. This file identifier will represent the fact that a particular file does not reside in a

particular directory. Subsequent searches for this same pair (e.g., Directory/Filename) are

5   quickly satisfied from the negative name cache. When the negative name cache fills up, the

present invention in an embodiment uses a random replacement policy in which a random

entry in the cache is thrown out to make room for the new entry just created. Additionally,

when a file is created, UFM needs to remove a corresponding entry from the negative name

cache if such an entry exists.

10

        The foregoing description and associated figures detail only illustrative examples of

the environment in which the invention can be used and are not intended to be limiting. In

addition, various programming languages, software platforms, operating systems, hardware

components, memory components, and other technology may utilize the negative cache

15   system and method to provide more efficient and reliable means for accessing data in storage

devices. Furthermore, current technology can always be enhanced to incorporate the most

advanced available technology to implement the negative cache system and method.

Variations and modifications of the present invention is apparent to one skilled in the art, and

the above disclosure is intended to cover all such modifications and equivalents.

20